DOCUMENT RESUME

ED 270 077 IR 012 076

AUTHOR McMeen, George R.
TITLE Designing and Creating Computer-Assisted Instruction.
PUB DATE 17 Oct 85
NOTE 26p.; Paper presented at the Annual Conference of the Association for California College Tutorial and Learning Assistance (12th, Sacramento, CA, October 17, 1985).
PUB TYPE Guides - Non-Classroom Use (055) -- Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Authoring Aids (Programing); *Computer Assisted Instruction; Computer Software; Flow Charts; *Instructional Design; Instructional Innovation; *Microcomputers; *Programing; *Programing Languages; Teaching Methods
IDENTIFIERS *Pilot Author Language

ABSTRACT

Designed to encourage the use of a defined methodology and careful planning in creating computer-assisted instructional programs, this paper describes the instructional design process, compares computer-assisted instruction (CAI) and programmed instruction (PI), and discusses pragmatic concerns in computer programming. Topics addressed include: instructional goals and objectives; task description and flow charting; similarities and points of tangency between CAI and PI; linear and branching programming techniques; categories of CAI programs (drill and practice, tutorial, simulations, and games); writing a CAI program; authoring tools such as AUTHOR, ApplePILOT and SuperPILOT; and PILOT commands. Two examples of short PILOT programs are provided. It is pointed out that today's increase in microcomputer sales requires educators to recognize the importance of good programming in courseware/software; it is also important for teachers to be able to make use of corresponding developments in the instructional application of microcomputers in the classroom and in the utilization of authoring tools and simple methods of programming. A list of references is provided as well as appendices containing sample computer programs for a math drill and practice problem, a tutorial, and a program involving low-resolution graphics. (JB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
OERI
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
[X] This document has been reproduced as received from the person or organization originating it.
[ ] Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official position or policy.

ED270077

Designing and Creating Computer-Assisted Instruction

A Paper Presented at the
12th Annual Conference
of the
Association for California College Tutorial and Learning Assistance
Sacramento, California
October 17, 1985

George R. McMeen
Director of the Learning & Resource Center
College of Education
University of Nevada Reno
Reno, NV 89557

IR012076

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

George R. McMeen

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Table of Contents

	Page
Introduction	1
Instructional Goals and Objectives	1
Task Description and Flow Charting	2
A Comparison	4
Programming	7
Writing a CAI Program	10
Authoring Tools	11
Creating a Program with AUTHOR	13
Creating a Program with Apple PILOT and SuperPILOT	14
Conclusion	16
References	17
Appendix A	18
Math Drill-and-Practice Program	19
Tutorial Example	21
Program Involving Low-Res Graphics	23

Figures

1. Standard Symbols	2
2. Task Involving Adding 2-Digit Numbers	3

Introduction

The design and creation of computer-assisted instructional programs may be accomplished through the use of a defined methodology and careful planning and organization of instructional material. This paper describes the instructional design process, provides a comparison between computer-assisted instruction and programmed instruction, and relates pragmatic concerns in programming.

Instructional Goals and Objectives

Instructional design is a relatively structured approach which may be utilized in producing a successful computer-assisted instructional program. It involves stating goals and objectives. You must clearly and carefully think about what you want the program to do. If goals are not clearly stated, you may lose track of what is being taught. A goal can usually be stated as the answer to the question: What is it that the student is expected to do after the program ends? Thus the instructional goal may focus on what effect the program will have after it has been completed. More specifically, an instructional objective will focus upon learner activity. For example, an instructional goal might be: Upon completion of this program, the learner will be able to multiply random two-digit numbers with 90% accuracy. This instructional objective also includes the conditions: numbers to be multiplied will be randomly drawn and they will be two-digit numbers, as well as the standard of acceptable performance--90% accuracy. This is really a behavioral objective. Such objectives are useful in stating what it is that you expect your program to be able to do and what it is that you expect students to do. From this beginning, it becomes easier to develop a task description and flow chart.

Task Description and Flow Charting

A process called "task description" is used to determine the actual content of a computer-assisted instructional program. Here an objective is carefully analyzed to determine how it may be accomplished. Sequencing of steps or tasks to attain an objective are of particular concern, and priority may be given to certain steps. Individual steps or tasks are then arranged in some orderly fashion.

Linear tasks are active tasks that require the same sequence of all learners, although alternate routes may be possible. One common way of preparing a task description of a linear task is to flow chart it. Flow charting can also be used with other kinds of tasks such as variable sequences or cognitive tasks. Flow charting utilizes a set of standard symbols (Figure 1) to show graphically the individual components or steps of a process.

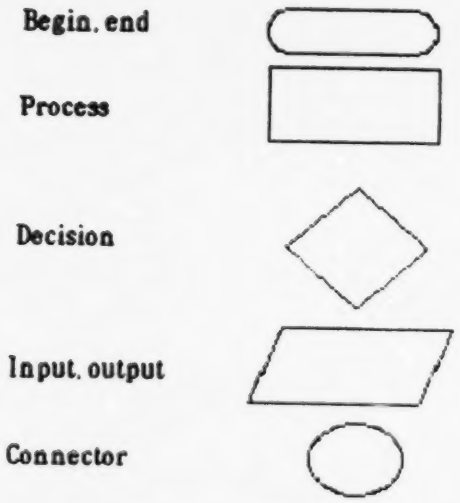

Fig. 1
Standard Symbols

These steps may be used to represent the component steps involved in a process leading to the atainment of a specific task such as adding 2-digit numbers (Figure 2).

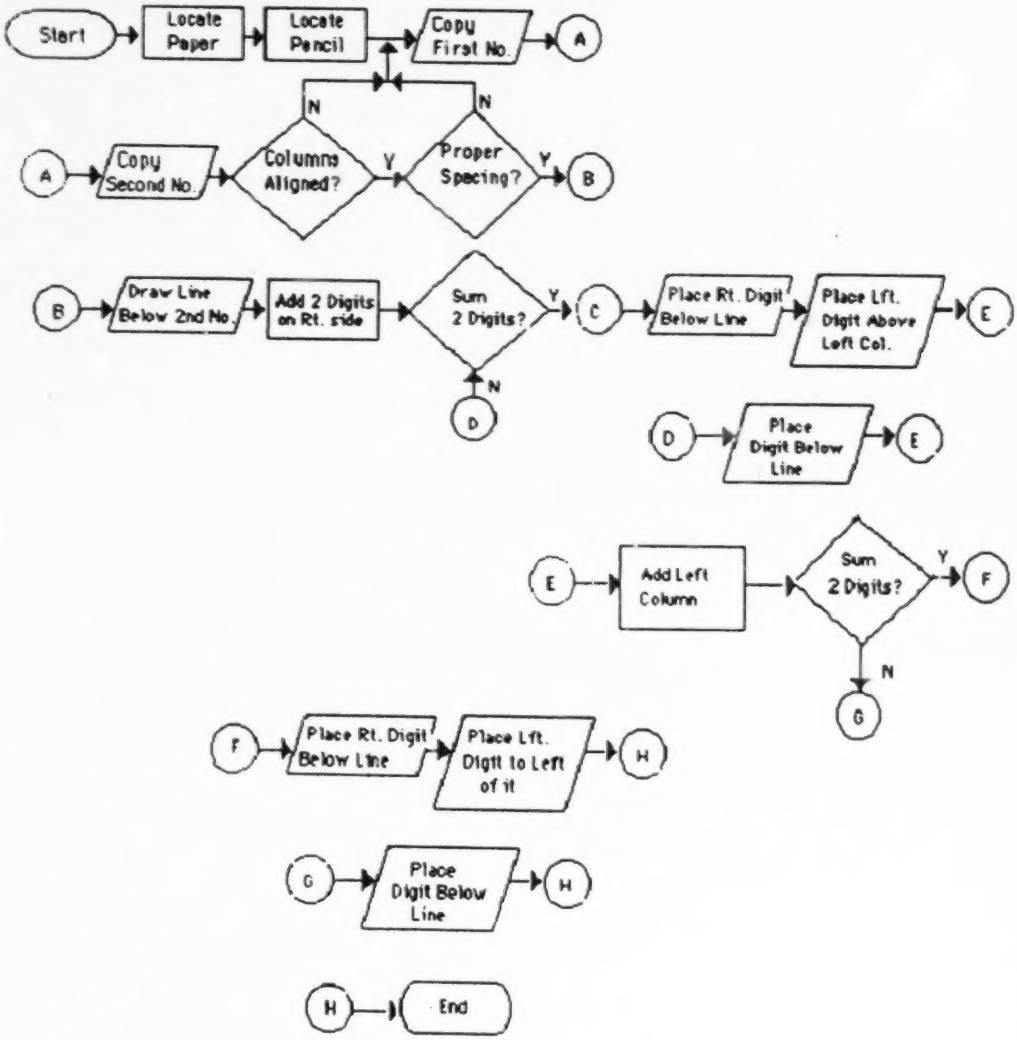

Fig. 2
Task Involving Adding 2-Digit Numbers

BEST COPY AVAILABLE

In writing a computer-assisted instructional program, the representation of the component steps leading to the attainment of a task is a very important stage of development, for the flow chart completes an "algorithm," or thought process in outling the steps to the completion of a certain task. If this is all there is to writing the program, one might begin writing the actual computer programming steps, using a specific programming language. However, there are usually many such algorithms in a program, and one should complete these algorithms before proceeding with the use of a programming language. Otherwise, careful organization of the program will likely be lost, and the work of programming will likely proceed by trial-and-error, taking even more time.

A Comparison

In many ways, clear trends may be seen between computer-assisted instruction (CAI) and programmed instruction (PI), to the extent that CAI may be viewed as the logical offshoot of programmed instruction. One obvious point of congruence concerns inter- actions between teachers, students, course materials, methods, and computers. Stevens (1981) views these as part of CAI, for the same ingredients that were part of programmed instruction are present in computer-assisted instruction. In fact, the design of computer-assisted instructional programs follows the instructional design patterns and includes many of the same variables that were part of programmed instruction in the 1960's. This is important: we should see the similarities and points of tangency between CAI and PI to understand more fully the development of computer-assisted instruction and what may be inherent in good programming techniques. For example, concern for individualized learning, self-pacing, immediate knowledge of the correctness of response and reinforcement are just as important to CAI as they were to PI. Taking stock of the methodologies presently being employed and studying their relationship to earlier methods is a key to developing improved programs that build upon past experience and have a

direction consistent with proven programming principles.

In viewing these two methods of providing instruction, the question of what is a teaching machine should be stated. Very simply, a teaching machine may be any of a variety of methods of presenting programmed materials. These methods can range from programmed textbooks to microcomputers. Either the textbook or the microcomputer can present a unit of verbal or symbolic information, provide a means of responding, and offer immediate knowledge of results, informing the student of the correctness of a response. Reinforcement to learning was immediate.

According to Holland (1960), a teaching machine was basically a device that provided an explicit response after exposing a small problem and then giving immediate confirmation about the correctness of the response. The "machinery" could be very simple, encompassing a range of things from a paper and pencil to a computer. With a textbook or a microcomputer, students would be able to progress at their own rates of learning. Programmed instruction was simply a system in which a logical sequence of steps leading to a specific goal was present. The individual learner read and answered at his/her own rate, while immediate knowledge of results was provided. According to Rigney and Fry (1961), the teaching-machine program had little to do with the control of a teaching machine. It simply presented material to the learner, exposing stimulus material, requiring some overt or covert response, and providing some form of knowledge of results. Seen in the context of learning theory, these aspects should be considered relevant to computers and other forms of educational technology today.

Three distinct historical influences may be seen in the literature about teaching machines and programmed instruction. These influences include Sidney L. Pressey, B.F. Skinner, and Norman A. Crowder.

The first patent for a teaching machine occurred in 1866. However, activity did not really begin until the 1920's when Sidney Pressey and his associates considered the idea of mechanized teaching. Pressey's work with multiple-choice questions keyed to a punchboard providing knowledge of results attracted the attention of later behavioral psychologists such as Porter and Skinner in the late 1950's. Educational psychologists have had differing views of the best method for providing knowledge of results to the learner. Some have emphasized the importance of the learner's making a physical or overt response, which provided behavioral feedback. According to Kaess and Zeaman (1960), while some went so far as to conclude that any device requiring no observable motor action would be merely a teaching aid since it would need to be supplemented by the teacher to gain effectiveness. Emphasis would be upon a physical response. Each step in programming would need to be small enough to produce correct responses with a positive frequency rate, and thereby bring the learner gradually closer to competent behavior. Pressey's use of multiple-choice material differed from that of other programmers because it contained plausible wrong responses. However, wrong responses might strengthen unwanted responses (Kaess and Zeaman, 1960). Skinner and others believed that precise control should be maintained over the pattern of responding in order to shape desired responses through successive approximation and reinforcement (Goldbeck and Briggs, 1960). Even Crowder thought that a certain level of error might be useful in intrinsic (branching) programming. To Crowder, automatic tutoring by intrinsic programming was an individually-used, instructor-less method of teaching. Skinner advocated teaching machines as a replacement to traditional instructional methods, while Pressey viewed them as devices to supplement instruction. Pressey and Crowder permitted some error in their programs, although Skinner believed that error levels should be reduced to the barest minimum.

Programming can avoid excessive repetition which has long been

regarded as essential in associative learning situations. How much repetition should there be in lessons? The answer is not clear and is still an optional choice for the instructional designer of a program. In paired associate and operant conditioning, repetition is a factor in forming associations through a process of gradual strengthening. In learning lists of items, the strength of association between each pair of items will develop gradually, with each repetition adding an increment to that bond, until it is so strong that the first item produces recall of the second (Rock, 1957). Only a limited number of associations can be formed in initial student learning. Thus some repetition is essential, although some associations may be formed in one trial, as shown by typical behavior in rote learning experiments. In this regard, Rock's (1957) observation that repetition may be an artifact of working with long lists of items seems appropriate.

Error levels in either linear or branching programs refer to the number of wrong responses that may possibly be made by the learner in answering items. Unexpected errors are generally believed to indicate deficiencies in a program (Holland, 1959). Galanter (1959) believed that error probability in verbal or symbolic material in a linear program should be low (less than 10 percent). Consequently, it was held that the number of errors in a linear program should be reduced to a minimum, because the linear program would be built around the idea of positive reinforcement of correct responses. Branching programming does not hold to a specific error probability, but high error levels would suggest possible internal problems within a program. Both forms of programming may be found in many computer-assisted instruction programs.

Programming

Two forms of programming content involve linear and branching techniques. Linear programming presupposes that people learn in terms of

small, discrete units. In this form of programming, the learner must make discrete responses. Each item is part of a logical sequence of frames steadily progressing toward some objective. The type of organization inherent in this is called a fixed sequence. In order to insure the effectiveness of the program, a minimum number of probable errors is present in this sequence (Holland, 1960).

A branching program, sometimes called intrinsic or scrambled, is based on the assumption that individuals learn from a variety of methods. Here teaching is a communicating process, and human learning takes place in many ways depending on the abilities and knowledge of the students, the nature of the subject matter, and other factors. The intrinsic programmer will not presume to know how to set up conditions under which efficient learning will inevitably occur. Instead, the programmer will use feedback to control the communication process (Rigney and Fry, 1961). The approach of this program is somewhat like a textbook: after presenting a unit of material in familiar textbook format, the program will test what has been learned by offering a question, *e.g.* a multiple-choice question. Feedback will provide immediate knowledge of results. The student will then branch to explanatory material, to several successive steps constituting a review, or to a remedial loop. Branching items are problems in a scrambled sequence that return the learner to previous items for remedial work or to advance ahead, skipping intervening steps. Intrinsic programs are necessarily branching. The reason for this is that a learner's response will affect progression in a sequence of items. If student performance falls below a certain level, the student may be directed to a branching set of items that provide remedial work (Coulson and Silberman, 1961, p. 9). In intrinsic programming, the proper construction of the right sequence of steps is important, for the learner's response to each step must be the result of learning the critical information which has been conveyed in the steps.

Other alternatives might include a combination of linear and branching programming techniques. These techniques can be combined in a style of programming called adaptive programming, which emphasizes the strengths of both. However, the two strong strains of programming remain. They are linear and branching programming, although they can be blended or combined in adaptive or hybrid styles.

The design and creation of computer-assisted instructional programs may be accomplished through careful planning and organization of instructional material. CAI programs may be grouped into the following categories:

1. Drill-and-practice. This form of courseware is aimed at reinforcing basic skills. Characteristically, a finite number of questions and a tally of the student's right and wrong answers are usually displayed. In some drill-and-practice programs, the teacher has the ability of determining the number and nature of the questions, diagnose a student's errors and provide extra practice. Record-keeping functions may be found in some drill-and-practice programs as a part of feedback to the student about the correctness of responses.

2. Tutorial. Tutorial programs teach concepts to students, and they assume little or no prior student knowledge of the subject matter involved. They carefully guide learners through all the steps required to understand the concepts being taught. Words and/or computer graphics, and possibly some branching will be present.

3. Simulations. These programs model real-life or experimental situations that might otherwise be difficult to produce in the classroom.

4. Games. Games can be an interesting means of instruction in situations where there may be a careful match between gaming functions and instructional objectives.

Perhaps the easiest of these courseware formats to develop is the tutorial, which may largely consist of printed statements and possibly graphic displays. Here, a branching arrangement might likely be present for sequences of information-- advanced and remedial loops might send the

student forward or backward for additional information.

Writing a CAI Program

Writing a computer-assisted instructional program on your own involves time and a knowledge of simple programming principles in a programming language such as APPLESOFT BASIC, a variation of BASIC for the Apple II computer. The program seen in this context becomes a sequence of instructions that informs the computer about the tasks that need to be performed to carry out the purpose of the program. Three programming sequences may be seen in Appendix A. The first of these is a math program written in APPLESOFT BASIC, which offers drill-and-practice problems in addition, subtraction, multiplication and division and permits the student to determine the number of problems chosen at random. It is relatively clear-to-the-point involving mathematical equations in subroutines to determine the correctness of a response to a question posed as the result of the mathematical operation which has been selected. In fact, it is very simple to create, and in simplicity there is a certain beauty. Unfortunately, this kind of beauty is lost in the second example, which-- although it is a very abbreviated example of what may be actually involved in a tutorial-- is very lengthy and obviously laborious to write. This is an interesting observation because in this program we really only have one frame and a multiple- choice question with four possible answers. The student would read the content provided in numerous PRINT statements and proceed to remedial loops or advance to Frame 2 (which exists only as a title). The length of this program for only one frame in a branching program suggests the need for seeking an authoring tool to simplify things. Nevertheless, for the ambitious educator-programmers, this may be the kernel of developing a tutorial program.

Both of the first two examples involve a cursor movement command

(HOME in APPLESOFT BASIC), as well as FOR...NEXT loops for time delays.

Neither example, however, shows how graphics might be interwoven into a program. Thus the third example provides a very simple way of incorporating Low-Res graphics in a program by using a Low-Res editor adapted from a fairly well known example by Williams, Kernaghan, & Kernaghan (1983). This makes use of "PRINT CHR$(4)"BLOAD FLAG" to load a low-res graphic of the American flag, which has been created using Williams, Kernaghan, & Kernaghan's Low-Res editor with a slight modification by the presenter. Other designs could be created using this editor and saved under other names with the BSAVE command, which has been added to the editor as an important modification.

These examples are provided to suggest what might be involved for the beginner in developing either a drill-and-practice program or a tutorial. The second example reinforces the obvious need for using some authoring program to assist in writing a tutorial.

Authoring Tools

Another alternative to programming centers about the use of authoring applications software. A program such as AUTHOR may be used to create CAI. AUTHOR reviews easy-to-use menus and screen cue prompts. The value of this applications program is that it requires little knowledge of programming or computer commands, yet it is easy to use. Lessons are self-instructional, self-paced and interactive.

Authoring programs usually lead to satisfying results in developing computer-assisted instructional programs after only a few tries and a few hours of effort. A variety of such applications programs exist. They include tools such as Apple PILOT and Apple SuperPILOT, which use a computer language named PILOT to help educators write instructional

programs (PILOT means Programmed Inquiry, Learning or Teaching). They are particularly useful in that they include four editors (lesson text, character set, graphics and sound effects), which assist in programming. Generally, authoring tools have certain features in common: they allow for author input in determining frame settings in which key attributes and directions may be specifically written to guide the student through a sequence of steps. The result is the creation of on-screen computer lessons.

These software applications for authoring presume no computer programming knowledge necessary to author lessons. They do assume the need for developing self-instructional, self-paced and interactive lessons, and interest in branching instruction. Quick and successful use of most authoring systems can be attained without programming skills. Authoring progress can be attained by following simple screen prompts that assist in choosing subsequent blocks of instruction or permit reviewing lessons. Thus lesson design becomes relatively quick and easy.

In a self-instructional, self-paced, interactive mode, students have opportunities to learn at their own rates of learning. Thus they are able to pace their own instruction. This leads to individual mastery of the material at the learner's own rate of learning. Lessons designed with this in mind become engaging and reinforcing because of the use of frequent interactive questions and responses. Self-instructional, self-paced, interactive lessons may be used to build interest and motivation.

A feature of lessons that have been developed by the use of authoring systems is that a learner can be branched to other instructional sequences. Thus each question may be used diagnostically in determining whether the learner needs additional instruction to improve understanding. In addition, the instructional designer may decide to branch the learner to remedial or advance instruction if an incorrect or correct answer is selected.

Creating a Program with AUTHOR

The entire AUTHOR system consists of (a) a learning program and (b) an authoring program. Up to 105 frames can be created for each lesson package. One flexible disk must be reserved for each lesson diskette. Lessons may be created using multiple-choice, fill-in blanks (completion), sequencing, and true/false.

Instructional paths may be created in two ways: "initial" and "review" modes. Initital permits the learner to progress through each lesson from start to finish. The review mode allows the learner to advance to a review menu where specific lesson elements can be reviewed. Thus the learner can progress through the lesson at his/her own rate or move directly to sections for advance work or remediation.

Throughout the program, records are kept for each learner to reflect student progress. Records may be kept for appoximately 200 students working for any particular lesson package.

One disadvantage is the lack of graphics or sound. However, in the development of a straight tutorial program, the programmer and the teacher involved may not have a need for graphics or sound. The real need may be for simply delivering instructional content without any embellishment. In such a case, AUTHOR would be satisfactory. However, tools such as Apple PILOT and Apple SuperPILOT provide more attractive options for creating a more motivating program.

Creating a Program with Apple PILOT and SuperPILOT

Apple PILOT and Apple SuperPILOT are useful and provide a simple way of programming and obviating many of the programming steps shown

in the tutorial in Appendix A. With these tools, the programmer may use the lesson text editor to write new programs, edit existing ones, run programs from author mode, and print lessons; the character set editor to create a unique set of characters or symbols; the graphics editor to make pictures that can be included in lessons; the sound effects editor to create sound effects and music to accompany lessons.

PILOT commands consist of three elements: (a) instruction name, (b) a colon (:), and (c) instruction "object." In the instruction name, a single (T) means "type text," and it may be used to enter 250 characters in lines of information in the same way that a PRINT command could be used in BASIC. The text editor automatically formats your instruction to fit the 40 character screen. (A:) represents the "accept" instruction, which informs the computer to wait for a response and store that response in memory. The "object" of an accept instruction (A:) is optional; it identifies a location in the computer's memory where the student's response is to be stored. The computer automatically stores the response in a temporary storage area called the "answer buffer." (AS:) may be used to accept a single-character response (the user does not need to touch a return key). (M:) means "match." This symbol is used to determine whether the response typed by the student is the one wanted. Correct use of this symbol involves telling the computer what response(s) are anticipated. The object field of a (M:) match instruction contains the items-- correct or incorrect-- that you would like the computer to match against. Items listed in the object field following (M:) are compared with the current contents of the answer buffer to determine whether a "match" is present. Branching may be accomplished using (J:) for "jump." In a branching item, remediation may be allowed until the answer and the lesson are correct. When a (J:) instruction is executed, the computer forgets where it was in the program and begins at the labeled destination shown in the (J:) instruction. Subroutines may be created using (U:) to direct the program to a subroutine or section of the program and then return to the main program where

this subroutine began. The subroutine does not have to be written more than once. When the subroutine is finished, the program will return to the point where the branch occurred.

Two examples are provided below of short PILOT programs. The first example involves a (J:) command to branch to another line (prob1). This is analogous to a GOTO command in BASIC. Remember that "Y" stands for "Yes, correct" and "N" for "No, incorrect" conditions in commands such as **TY1, TY2** and **JN1** (1 and 2 represent first and second tries or times). The second example shows the use of (U:) to branch to a subroutine (similar to GOSUB). (E:) terminates a subroutine (similar to RETURN).

```
*prob 1
T:How much is 2 + 2?
A:
M:4
TY1:Good, you know your addition!
TY2:That's more like it!
TN1:Think carefully. Try again.
TN2:Ask your teacher for help.
JN1:prob2
```

```
*prob2
T:Guess a number:
U:range
A:
M:5!five

*range
T:It is between 1 and 100
E:
```

The second example in the preceding programs consists of a main program called "prob2" and a subroutine named "range." This latter example should be regarded as a program separate from the preceding one.

Conclusion

There is a great deal of excitement and interest attending the microcomputer revolution in education. The key question is no longer whether or not this new technology will be used, but how it will be developed and utilized. Although computer scientists have been experimenting with computers and developing educational applications for at least two decades, the emergence of microcomputers in education is a relatively new development beginning in the late 1970's. With the increase in microcomputers of all different kinds being sold, it is necessary for educators to recognize the importance of good programming in courseware/software today. Understanding the connection between CAI and PI is a means of identifying what might constitute worthwhile programming in computer-assisted instruction. It is likely that more dramatic and exciting developments in the microcomputer field will continue. It is also important for teachers to be able to make use of corresponding developments in the instructional application of micro-computers in the classroom and in the utilization of authoring tools and simple methods of programming involving short programs of a drill-and-practice nature, as in the case of mathematics, and the employment of low-res graphics developed with an editor and written into a program. Deciding how courses utilizing micro-technology will be programmed to meet instructional needs is a very important curricular challenge, and there are many opportunities for teachers and programmers to be creative and imaginative in their instructional use of various methods for writing programs.

References

Coulson, J.E., and Silberman, H.F. Automated Teaching and Individual Difference. AV Communication Review, 1961, 9(10), 5-15.

Galanter, E. The Ideal Teacher. In E. Galanter (Ed.), Automatic Teaching: The State of the Art. New York: Wiley, 1959.

Goldbeck, R.A., and Briggs, L.J. An Analysis of Response Mode and Feedback Factors in Automated Instruction. A Technical Report Prepared for the Office of Naval Research. Santa Barbara: American Institute for Research, 1960.

Holland, J.G. Practical Methods of Evaluating Teaching Machines and Programs. An Unpublished Extract Written at Harvard University, November, 1960.

Holland, J.G. Teaching Machines: An Application of Principles from the Laboratory. A Paper Presented at the Invitational Conference on Testing Problems Sponsored by the Educational Testing Service, October, 1959.

Kaess, W., and Zeaman, D. Positive and Negative Knowledge of Results on a Pressey-type Punchboard. Journal of Experimental Psychology, 1960, 60(1), 12-17.

Rigney, J.W., and Fry, E.B. Current Teaching-Machine Programs and Programming Techniques. AV Communication Review, 1961, 9(3), 7-119.

Rock, I. The Role of Repetition in Associative Learning. The American Journal of Psychology, 1957, 70, 186-193.

Stevens, D.J. Computers, Curriculum, and Careful Planning. Educational Technology, 1981, 21(11), 21-24.

Williams, K., Kernaghan, B., and Kernaghan, L. Apple II Graphics. New York: Brady, 1983.

Appendix A

Math Drill-and-Practice Problem

Tutorial Example

Program Involving Low-Res Graphics

Math Drill-and-Practice Problem

```
10 HOME : REM RANDOM MATH DRILL-AND-PRACTICE
20 DIM N$(25) : REM DIMENSION THE NAME VARIABLE
30 REM INTRODUCTION
40 INPUT "WHAT IS YOUR NAME";N$
50 PRINT : PRINT "THANKS, ";N$
60 C = 0 : REM VARIABLE FOR CORRECT ANSWER COUNT IS CLEARED HERE AFTER DRILL
70 W = 0 : REM VARIABLE FOR WRONG ANSWER COUNT IS CLEARED HERE AFTER DRILL
80 INPUT "HOW MANY MATH PROBLEMS DO YOU WANT TO DO";HM
90 REM : STUDENT CHOOSES KIND OF MATH PROBLEMS
100 PRINT : PRINT "WHAT KIND OF MATH PROBLEMS DO YOU WANT?"
110 PRINT
120 PRINT "   1 - ADD"
130 PRINT "   2 - SUBTRACT"
140 PRINT "   3 - MULTIPLY"
150 PRINT "   4 - DIVIDE"
160 PRINT : INPUT "   ENTER A NUMBER";K
170 HOME : REM SELECTION
180 IF K = 1 THEN GOSUB 360
190 IF K = 2 THEN GOSUB 480
200 IF K = 3 THEN GOSUB 610
210 IF K = 4 THEN GOSUB 740
220 IF K < 1 THEN 160
230 IF K > 4 THEN 160
240 HOME: REM REPORTING NUMBER CORRECT AND WRONG
250 PRINT N$;", YOU HAD ";C;" CORRECT ANSWER(S)"
260 PRINT "AND ";W;" WRONG." : FOR ZZ = 1 TO 1200:NEXT ZZ
270 HOME : REM DO MORE MATH OR EXIT
280 PRINT "DO YOU WANT TO--" : PRINT
290 PRINT "   A. DO MORE MATH"
300 PRINT "   B. EXIT" : PRINT
310 PRINT "       (ENTER 'A' OR 'B')
320 PRINT : INPUT Q$
330 PRINT
340 IF Q$ = "A" THEN 60
350 END
360 REM 'ADD' SUBROUTINE
370 FOR X = 1 TO HM
380 N1 = INT(5 * RND(1)) + 1
390 N2 = INT(5 * RND(1)) + 1
400 PRINT : PRINT N1;" + ";N2;" = "
410 INPUT R
420 A = N1 + N2
430 IF R = A THEN PRINT "GOOD" : C = C + 1
440 IF R <> A THEN PRINT "WRONG" : W = W + 1
450 FOR ZZ = 1 TO 1200 : NEXT ZZ : PRINT
460 NEXT X
470 RETURN
480 REM 'SUBTRACT' SUBROUTINE
490 FOR X = 1 TO HM
500 N1 = INT(5 * RND(1)) + 1
510 N2 = INT(5 * RND(1)) + 1
520 PRINT:PRINT N1;" - ";N2;" = "
530 INPUT R
540 S = N1 - N2
550 PRINT
560 IF R = S THEN PRINT "GOOD" : C = C + 1
570 IF R <> S THEN PRINT "SORRY" : W = W + 1
580 FOR ZZ = 1 TO 1200 : NEXT ZZ : PRINT
590 NEXT X
600 RETURN
610 REM 'MULTIPLY' SUBROUTINE
620 FOR X = 1 TO HM
630 N1 = INT(5 * RND(1)) + 1
640 N2 = INT(5 * RND(1)) + 1
650 PRINT : PRINT N1;" X ";N2;" = "
660 INPUT R
```

```
670 M = N1 * N2
680 PRINT
690 IF R = M THEN PRINT "GOOD" : C = C + 1
700 IF R <> M THEN PRINT "SORRY" : W = W + 1
710 FOR ZZ = 1 TO 1200 : NEXT ZZ : PRINT
720 NEXT X
730 RETURN
740 REM 'DIVIDE' SUBROUTINE
750 FOR X = 1 TO HH
760 N1 = INT(5 * RND(1) + 1
770 N2 = INT(5 * RND(1) + 1
780 PRINT : PRINT N1;" / ";N2;" = "
790 INPUT R
800 D = N1 * N2
810 PRINT
820 IF R = D THEN PRINT "GOOD" " C = C + 1
830 PRINT "SORRY" : W = W + 1
840 FOR ZZ = 1 TO 1200 : NEXT ZZ : PRINT
850 NEXT X
860 RETURN
```

Tutorial Example

```
10 REM TUTORIAL EXAMPLE
20 DIM N$(25)
30 C = 0 : REM COUNTER FOR CORRECT ANSWERS
40 W = 0 : REM COUNTER FOR WRONG ANSWERS
50 HOME
55 REM INTRODUCTION
60 PRINT "YOU ARE GOING TO LEARN HOW THE UNITED"
70 PRINT "STATES IS GOVERNED." : PRINT
80 REM STUDENT NAME
90 INPUT "ENTER YOUR NAME: ";N$
100 PRINT : HOME : REM FRAME 1
110 PRINT "THE FEDERAL GOVERNMENT EXERCISES"
120 PRINT "CONTROL OVER ALL THE STATES." : PRINT
130 PRINT "THE FEDERAL GOVERNMENT HAS AUTHORITY"
140 PRINT "GIVEN IT BY THE CONSTITUTION." : PRINT
150 PRINT "THE CONSTITUTION DEFINES THE"
160 PRINT "SYSTEM OF GOVERNMENT IN THE"
170 PRINT "UNITED STATES.  AS THE BASIC"
180 PRINT "LAW, THE CONSTITUTION OUTLINES"
190 PRINT "THE TASKS INVOLVED IN MAKING"
200 PRINT "AND ENFORCING THE LAWS WHICH"
210 PRINT "GOVERN THE UNITED STATES."
220 FOR ZZ = 1 TO 6000: NEXT ZZ
230 HOME
235 REM FRAME 1 MULTIPLE-CHOICE QUESTIONS
240 PRINT : PRINT "WHAT DEFINES THE SYSTEM OF"
250 PRINT "GOVERNMENT IN THE U.S.?" : PRINT
260 PRINT "  A. STATES"
270 PRINT "  B. CONSTITUTION"
280 PRINT "  C. LAWS"
290 PRINT "  D. FEDERAL GOVERNMENT" : PRINT
300 PRINT "ENTER THE LETTER FOR YOUR CHOICE."
310 INPUT A$
320 HOME
330 REM SELECTION
340 IF A$ = "A" THEN GOSUB 530
350 IF A$ = "B" THEN GOSUB 790
360 IF A$ = "C" THEN GOSUB 830
370 IF A$ = "D" THEN GOSUB 1120
380 HOME
390 REM FRAME 2 (INCOMPLETE)
400 REM HERE YOU WOULD BEGIN A SECOND QUESTION
410 REM IN YOUR TUTORIAL PROGRAM AND CONTINUE.
420 REM FOR THE SAKE OF BREVITY, THIS HAS BEEN
430 REM LEFT OUT EXCEPT FOR THE FOLLOWING TITLE
440 REM TO MARK WHAT WOULD BE THE BEGINNING OF
450 REM FRAME 2
460 PRINT : PRINT "FRAME 2 WOULD BEGIN HERE....."
470 FOR ZZ= 1 TO 3000: NEXT ZZ
480 REM REPORT CORRECT AND WRONG ANSWER TOTALS
490 HOME
500 PRINT N$;", YOU HAD ";C;" CORRECT"
510 PRINT "AND ";W;" WRONG."
520 END
530 PRINT "SORRY, ";N$;"."
540 PRINT
550 PRINT "THE CORRECT ANSWER WAS THE CONSTITUTION"
560 W = W + 1: REM WRONG ANSWER TALLY
570 FOR ZZ = 1 TO 4000 : NEXT ZZ
580 HOME
590 REM REMEDIATION
600 PRINT
610 PRINT "THE U.S. CONSTITUTION PROVIDES"
620 PRINT "A FEDERAL FRAMEWORK BY WHICH"
630 PRINT "THE STATES ARE GOVERNED."
```

```
640 FOR ZZ = 1 TO 3000 : NEXT ZZ
650 HOME
660 PRINT "WHAT PROVIDES THE FEDERAL FRAMEWORK"
670 PRINT "BY WHICH THE STATES ARE GOVERNED?" : PRINT
680 PRINT "  A. STATE LAWS"
690 PRINT "  B. FEDERAL LAWS"
700 PRINT "  C. CONSTITUTION"
710 PRINT "  D. ALL OF THESE" : PRINT
720 PRINT N$;", ENTER A LETTER"
730 PRINT "FOR YOUR CHOICE."
740 INPUT A1$
750 IF A1$ <> "C" THEN PRINT "TRY AGAIN."
760 FOR ZZ = 1 TO 3000 : NEXT ZZ
770 IF A1$ <> "C" THEN GOTO 740
780 RETURN
790 PRINT N$;", GOOD GOING."
800 C = C + 1 : REM CORRECT ANSWER TALLY
810 FOR ZZ = 1 TO 3000 : NEXT ZZ
820 RETURN
830 PRINT "SORRY, ";N$;"."
840 PRINT
850 PRINT "THE CORRECT ANSWER WAS THE CONSTITUTION."
860 FOR ZZ = 1 TO 3000 : NEXT ZZ
870 W = W + 1 : REM WRONG ANSWER TALLY : PRINT
880 PRINT "TRY THIS:"
890 FOR ZZ= 1 TO 2000: NEXT ZZ
895 HOME
900 REM REMEDIATION
910 PRINT "THERE ARE MANY LAWS, BUT THE"
920 PRINT "HIGHEST LAW OF THE LAND IS THE"
930 PRINT "U.S. CONSTITUTION."
940 PRINT
950 FOR ZZ = 1 TO 3000 : NEXT ZZ
960 HOME
970 PRINT "WHAT IS THE HIGHEST LAW OF THE LAND?"
980 PRINT
990 PRINT "  A. STATE LAWS"
1000 PRINT "  B. U.S. CONSTITUTION
1010 PRINT "  C. LOCAL LAWS"
1020 PRINT "  D. FEDERAL LAWS"
1030 PRINT
1040 PRINT "ENTER A LETTER FOR YOUR CHOICE."
1050 INPUT A2$
1060 IF A2$ <> "B" THEN PRINT "TRY AGAIN."
1070 FOR ZZ = 1 TO 3000: NEXT ZZ
1080 IF A2$ <> "B" THEN 960
1090 PRINT "GOOD.  LET'S CONTINUE."
1100 FOR ZZ = 1 TO 3000: NEXT ZZ
1110 RETURN
1120 PRINT "NO.  THE ANSWER IS THE CONSITUTION."
1130 HOME
1140 PRINT "TRY THIS:": FOR ZZ = 1 TO 1000 : NEXT ZZ
1150 HOME
1160 REM REMEDIATON
1170 PRINT "THE FEDERAL GOVERNMENT HAS AUTHORITY"
1180 PRINT "OVER THE STATES.  THIS AUTHORITY HAS"
1190 PRINT "GRANTED BY THE CONSTITUTION."
1200 HOME
1210 PRINT "WHAT GAVE THE FEDERAL GOVERNMENT ITS AUTHORITY?"
1220 PRINT
1230 PRINT "   A. CONSTITUTION"
1240 PRINT "   B. FEDERAL LAWS"
1250 PRINT "   C. STATE LAWS"
1260 PRINT "   D. LOCAL LAWS"
1270 INPUT A3$
1280 FOR ZZ = 1 TO 3000: NEXT ZZ
1290 IF A3$ <> "A" THEN PRINT "WRONG, TRY AGAIN."
1300 IF A3$ <> "A" THEN 1200
1310 RETURN
```

Program Involving Low-Res Graphics

```
10 HOME
20 REM FLAG DEMO
30 PRINT "YOU WILL SEE A FLAG"
40 PRINT : PRINT "TELL ME WHAT COUNTRY IT BELONGS TO"
50 FOR ZZ = 1 TO 6000 : NEXT ZZ
60 GR
70 PRINT CHR$(4)"BLOAD FLAG"
80 FOR ZZ = 1 TO 4000 : NEXT ZZ
90 TEXT
100 HOME
110 PRINT "WHAT COUNTRY?"
120 INPUT F$
130 IF F$ = "U.S." THEN 150
140 GOTO 110
150 PRINT "CORRECT"
160 END
```

*Note: The binary table called FLAG has been created using Low-Res Editor created by Williams, Kernaghan, & Kernaghan (1983).